OS Internals

User Application

C Library (libc)

System Call Interface

Kernel

Arch-dependent kernel code

Hardware Platform

User space

Kernel space
OS Internals

**System Call Interface**

- **File System Management**
- **Memory Management**
- **Process Management**
  - scheduler
  - IPC
  - synchronization
- **I/O Management (device drivers)**
- **Hardware Control (Interrupt handling, etc.)**

**Hardware**

**User space**
- shell
- ls
- ps

**Kernel space**
- trap
#include <stdio.h>
int main ()
{
    
    printf ("Greetings");
    
    return 0;
}
Protected Instructions

- Protected or privileged instructions
  - Direct I/O access
    - Use privileged instructions or memory-mapping
  - Memory state management
    - Page table updates, page table pointers
    - TLB loads, etc.
  - Setting special “mode bits”
  - Halt instruction
How does the processor know if a protected instruction should be executed?

- The architecture must support at least two modes of operation: kernel and user mode
  - 4 privilege levels in IA-32: Ring 0 > 1 > 2 > 3
- Mode is set by a status bit in a protected processor register
  - User programs in user mode, OS in kernel mode
  - Current Privilege Level (CPL) in IA-32: CS register
- Protected instructions can only be executed in the kernel mode
OS Protection

- Crossing protection boundaries
  - User programs must call an OS to do something privileged.
    - OS defines a sequence of system calls
  - There must be a system call instruction that:
    - causes an exception, which invokes a kernel handler
    - passes a parameter indicating which system call to invoke
    - saves caller’s state (registers, mode bits) so they can be restored
    - OS must verify caller’s parameters (e.g. pointers)
    - must provide a way to return to user mode when done.
    - (cf.) INT 0x80 in Linux
OS Protection

- Making a system call
  - System call changes mode to kernel
  - Return from system call resets it to user
OS Operation

- OS is basically interrupt-driven
  - Hardware interrupt
    - Timer
    - I/O
  - Software interrupt
    - System call
    - Exception
**Multiprogramming**

- **Batch system**
  - Job must wait for the preceding job to finish
  - One by one

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via job scheduling
  - When it has to wait (for I/O for example), OS switches to another job

- **Timesharing (multitasking)**
  - CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
  - Response time should be < 1 second
Memory Protection

- Requirements
  - OS must protect user programs from each other
    - Malicious users
  - OS must also protect itself from user programs
    - Integrity and security
Memory Protection

- Simplest scheme
  - Use base and limit registers
  - Base and limit registers are loaded by OS before starting a program
Memory Protection

- MMU (memory management unit)
  - Memory management hardware provides more sophisticated memory protection mechanisms
    - Base and limit registers
    - page table pointers, page protection, TLBs
    - virtual memory
    - segmentation
  - Manipulation of memory management hardware are protected (privileged) operations
Process Management

- A process is a program in execution
  - Program is a passive entity
  - Process is an active entity
- Process needs resources to accomplish its task
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one program counter specifying location of next instruction to execute
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
Synchronization

- Problems
  - Interrupt can occur at any time and may interfere with interrupted code
  - OS must be able to synchronize concurrent processes

- Synchronization
  - Turn off/on interrupts
  - Use a special atomic instructions
    - read-modify-write (e.g., INC, DEC)
    - test-and-set
    - LOCK prefix in IA32
    - LL (Load Locked) & SC (Store Conditional) in MIPS
I/O Subsystems

- One purpose of OS is to hide peculiarities of hardware devices from the user
- General device-driver interface
- Drivers for specific hardware devices
- I/O subsystem includes
  - Buffering (storing data temporarily while it is being transferred)
  - Caching (storing parts of data in faster storage for performance)
  - Spooling (the overlapping of output of one job with input of other jobs)
Storage Management

- registers
- cache
- main memory
- solid-state disk
- hard disk
- optical disk
- magnetic tapes
Storage Management

- **OS provides uniform, logical view of information storage**
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
  - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

- **File-System management**
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media
Design and Implementation of OS not “solvable”
- Some approaches have proven successful

Internal structure of different OSs can vary widely

Start design by defining goals and specifications

Affected by choice of hardware, type of system

User goals and System goals
- User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
Important principle to separate
- Policy: What will be done?
- Mechanism: How to do it?

Mechanisms determine how to do something, policies decide what will be done

Separation of policy from mechanism
- Allows maximum flexibility if policy decisions are to be changed later

Specifying and designing an OS is highly creative task of software engineering
MS-DOS – written to provide the most functionality in the least space
- Not divided into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
# Operating System Structures

## UNIX System

<table>
<thead>
<tr>
<th>(the users)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shells and commands</td>
</tr>
<tr>
<td>compilers and interpreters</td>
</tr>
<tr>
<td>system libraries</td>
</tr>
</tbody>
</table>

### system-call interface to the kernel

| signals |
| terminal handling |
| character I/O system |
| terminal drivers |

| file system |
| swapping block I/O system |
| disk and tape drivers |

| CPU scheduling |
| page replacement |
| demand paging |
| virtual memory |

### kernel interface to the hardware

| terminal controllers terminals |
| device controllers disks and tapes |
| memory controllers physical memory |
Layered Approach

- The operating system is divided into a number of layers
  - Each built on top of lower layers
  - Bottom layer is hardware
  - Highest is user interface
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
Microkernel VS Monolithic Kernel

- Moves as much from the kernel into user space
- Mach example of microkernel
  - Mac OS X kernel (Darwin) partly based on Mach
- Communication takes place between user modules using message passing

Benefits
- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
- More secure

Detriments
- Performance overhead of user space to kernel space communication
Microkernel System Structure

- Application Program
- File System
- Device Driver

Interprocess Communication
- messages
- CPU scheduling
- memory management

User mode
Kernel mode
Hardware

Microkernel
Many modern operating systems implement loadable kernel modules

- Uses object-oriented approach
- Each core component is separate
- Each talks to the others over known interfaces
- Each is loadable as needed within the kernel

Overall, similar to layers but with more flexible

- Linux, Solaris, etc
Module Approach in Solaris

- Device and bus drivers
- Scheduling classes
- File systems
- Loadable system calls
- Miscellaneous modules
- STREAMS modules
- Executable formats

Core Solaris kernel
### Mac OS X

<table>
<thead>
<tr>
<th>Graphical User Interface</th>
<th>Aqua</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Environments and Services</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>Cocoa</td>
</tr>
<tr>
<td>Kernel Environment</td>
<td></td>
</tr>
<tr>
<td>Mach</td>
<td>BSD</td>
</tr>
<tr>
<td>I/O Kit</td>
<td>Kernel Extensions</td>
</tr>
</tbody>
</table>